# 第一次编程练习报告

姓名：周钰宸 学号：2111408 班级：信安一班

##### **编程练习1——使用Eratosthenes筛法打印1 000 000内所有素数及个数**

* **源码部分：**

#include<iostream>

#include<cmath>

using namespace std;

bool if\_vegetable(int n) {//判断是否为素数

if (n <= 1)

return false;

else {

bool flag = true;

for (int i = 2; i < n; i++) {

if (n % i == 0)

flag = false;

}

return flag;

}

}

void Eratosthenes(int n) {

bool\* a = new bool[n];//下标是否是素数的bool类型数组

int count = 0;//素数个数

for (int i = 2; i < n; i++) {

a[i] = true;

}

for (int i = 2; i < sqrt(n); i++) {

if (if\_vegetable(i)) {//是素数

for (int j = 2; i \* j <= n; j++) {//去掉其所有整数倍数，设置为false

a[i \* j] = false;

}

}

}

for (int i = 2; i < n; i++) {

if (a[i]) {

cout <<i<< " ";

count++;

}

}

cout << endl;

cout << "Total:" << count << endl;

return;

}

int main() {

int n;

cin >> n;

Eratosthenes(n);

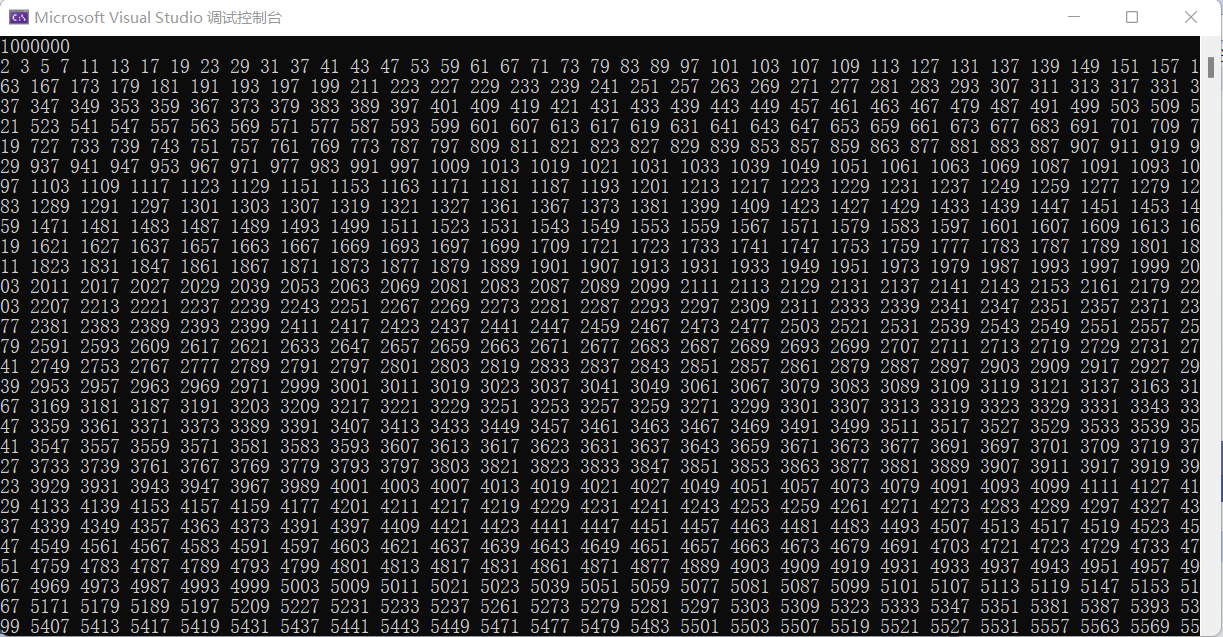
return 0;

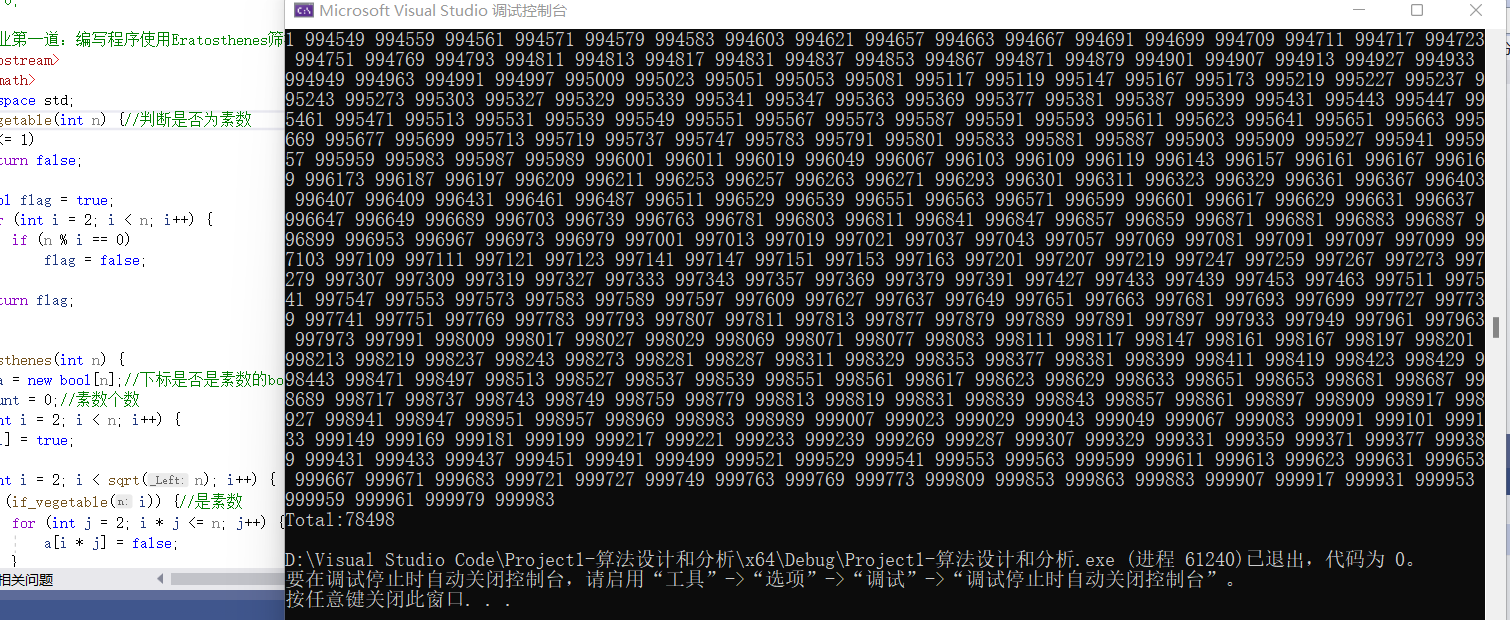
}

* **说明部分：**

1. 先编写一个函数if\_vegetable（忽略我的起名恶趣味咳咳），用来判断是否是素数
2. 在Eratosthenes函数中，定于一个bool类型的数组a判断对应下标位置的数是否为素数（下标从2开始才有意义）
3. 然后在Eratosthenes函数中按照Eratosthenes筛法依次进行，先求出不大于根号x的全部素数，只要发现一个，就把不超过x的该素数的大于1整数倍数作为下标位置的a设为false，即为合数。
4. 最后遍历a，输出所有素数。

* **运行示例：**





* **其他：思考问题——**

1. 对比筛法与普通算法的性能差异：

答：普通筛法是要遍历2-n的所有数字，然后依次判断是否为素数，而判断是否为素数的函数应该都与我编写的if\_vegetable函数类似，在调用这个函数时，这个函数本身也要在其内部遍历2-n的全部数字。因此存在循环的嵌套，普通筛法的时间复杂度为。而该筛法的外部遍历只需要从2-，当n很大时（数量级较高时），你与的数量级相差也会较大，因此通过将外部循环的数量级降低，该筛法的时间复杂度会低于，甚至接近于。因此该筛法对比普通算法，能够大大降低时间复杂度，优化性能。

1. 递归调用该算法求更大范围素数进行优化：

答：为了筛选出不大于n的所有素数，可以调用先筛选出不大于的所有素数，即Eratosthenes(）然后再依次调用递归进行，最后在素数为2和3的时候返回。

1. 求更大的素数（如数量级）该方法是否适用？ 会引入哪些新的问题？

答：不再适用。因为我使用的bool类型数组的最大长度为int类型，而int类型的最大值为0x7fffffff，是，因此当素数为超过这个数（如数量级）的数时，会超过int类型的范围，从而无法处理。可以考虑将int换为long或long long类型以求更大素数。

##### **编程练习2——编写程序计算最大公因数和最小公倍数**

* **源码部分：**

#include<iostream>

using namespace std;

int gcd(int a, int b) {//最大公因数，采用辗转相除法

int max, min;

if (a > b)

max = a, min = b;

else

max = b, min = a;

while (max % min != 0) {//只要余数不为0，就继续

int r = max % min;

max = min;

min = r;

}

return min;//此时的min（除数）是上一步的余数r，即最后一个不为0的余数，即最大公因数

}

int lcm(int a, int b) {//最小公倍数

return a \* b / gcd(a, b);

}

int main() {

int a, b;

cout << "a=";

cin >> a;

cout << "b=";

cin >> b;

cout<<"gcd(a,b)="<<gcd(a, b)<<endl;

cout<<"lcm(a,b)="<<lcm(a, b)<<endl;

return 0;

}

* **说明部分：**

1. 求最大公因数采用辗转相除法，求最后一个不为0的余数
2. 求最小公倍数，利用最小公因数和两个数乘积，由定理直接计算

* **运行示例：**



##### **编程练习3——编写程序实现算术基本定理**

* **源码部分：**

#include<iostream>

using namespace std;

bool if\_vegetable(int n) {//判断是否为素数

if (n <= 1)

return false;

else {

bool flag = true;

for (int i = 2; i < n; i++) {

if (n % i == 0)

flag = false;

}

return flag;

}

}

void Fundamental\_Theorem\_of\_Arithmetic(int n) {

int\* factor = new int[n];//存放因数

int NumOfFactor = 0;

int\* exponent = new int[n];//存放指数

int i = 2;//从最小的素数开始找起

for (int i = 0; i < n; i++) {

factor[i] = exponent[i] = -1;

}

if (if\_vegetable(n)) {//若n本身就为素数，直接输出

cout << n << "=" << n << "^1" << endl;

return;

}

int temp = n;

while (!if\_vegetable(temp)) {//只要此时的n（temp）没有被除的只剩下一个素数，就继续除

if (temp % i== 0) {//若能被这个素数整除，则该素数为其一个素数幂

factor[NumOfFactor] = i;//保存该素数幂

int current\_exponent = 1;//指数为1

temp = temp / i;

while (temp % i == 0) {//只要还能被i整除，就继续循环，增加指数

current\_exponent++;

temp = temp / i;

}

exponent[NumOfFactor] = current\_exponent;//保存指数

NumOfFactor++;

}

for (int j = i+1;;j++) {//若i不能整除n，则证明i不是其素数幂，去寻找大于i的最小素数，重复此过程

if (if\_vegetable(j))

{

i = j;

break;

}

}

}

bool flag = false;//是否最后剩下的temp是之前出现过的因子

for (int t = 0; t < NumOfFactor; t++) {

if (factor[t] == temp) {///若之前出现过

exponent[t]++;

flag = true;

break;

}

}

if (flag == false) {//若之前所剩下的n是未出现过的指数

factor[NumOfFactor] = temp;

exponent[NumOfFactor] = 1;

NumOfFactor++;

}

cout << n<<"=";

for (int k = 0; k < NumOfFactor; k++) {

if (k != NumOfFactor - 1)

cout << factor[k] << "^" << exponent[k] << "\*";

else

cout << factor[k] << "^" << exponent[k];

}

return;

}

int main() {

int n;

cout << "Please input n(n>0):";

cin >> n;

Fundamental\_Theorem\_of\_Arithmetic(n);

return 0;

}

* **说明部分：**

1. 同样设计一个函数if\_vegetable判断是否为素数
2. 算术基本定理求标准分解式的函数具体实现详见代码注释，这里只说明大体思路：从最小的素数（2）开始，依次用n除以该素数，若能被整除，则其为素数幂因子，将其和其指数保存；若不能被整除，寻找大于该素数的最小素数继续重复该操作，直至n也被除的变成素数了。
3. 最后别忘了把n最后变成的素数加入到因子中。

* **运行示例：**

